한국통신학회논문지 '12-09 Vol.37B No.09
http://dx.doi.org/10.7840/kics.2012.37B.9.795

MPI 브로드캐스트 통신을 위한 서킷 스위칭 기반의 파이프라인 체인 알고리즘 설계

윤 희 준◆, 정 원 영*, 이 용 석°

A Design of Pipeline Chain Algorithm Based on Circuit Switching for MPI Broadcast Communication System

Heejun Yun◆, Wonyoung Chung*, Yong-surk Lee°

요 약

본 논문에서는 분산 메모리 아키텍처를 사용하는 멀티프로세서에서 가장 병목 현상이 심한 집합통신 중 브로드캐스트를 위한 알고리즘 및 하드웨어 구조를 제안한다. 기존 시스템의 파이프라인 브로드캐스트 알고리즘은 전송 대역폭을 최대로 활용하는 알고리즘 이다. 하지만 파이프라인 브로드캐스트는 데이터를 여러 조각으로 나누어서 전송하기 때문에, 불필요한 동기화 과정이 반복된다. 본 논문에서는 동기화 과정의 중복이 없는 서킷 스위칭 기반의 파이프라인 체인 알고리즘을 위한 MPI 유닛을 설계하였고, 이를 systemC를 통하여 모델링하여 평가하였다. 그 결과 파이프라인 브로드캐스트 알고리즘과 비교하여 브로드캐스트 통신의 성능을 최대 3.3배 향상 시켰고, 이는 통신 버스의 전송대역폭을 거의 최대로 사용하였다. 그 후 verilogHDL로 하드웨어를 설계하였고, Synopsys사의 Design Compiler를 사용하여 TSMC 0.18 공정 라이브러리에서 합성하였으며 칩으로 제작하였다. 합성결과 제안하는 구조를 위한 하드웨어는 4,700 게이트(2-input NAND gate) 면적으로, 전체 면적에서 2.4%을 차지하였다. 이는 제안하는 구조가 작은 면적으로 MPSoC의 전체적인 성능을 높이는데 유용하다.

Key Words : MPSoC, Message Passing, Distributed memory, Broadcast Collective operation

ABSTRACT

This paper proposes an algorithm and a hardware architecture for a broadcast communication which has the worst bottleneck among multiprocessor using distributed memory architectures. In conventional system, The pipelined broadcast algorithm is an algorithm which takes advantage of maximum bandwidth of communication bus. But unnecessary synchronization process are repeated, because the pipelined broadcast sends the data divided into many parts. In this paper, the MPI unit for pipeline chain algorithm based on circuit switching removing the redundancy of synchronization process was designed, the proposed architecture was evaluated by modeling it with systemC. Consequently, the performance of the proposed architecture was highly improved for broadcast communication up to 3.3 times that of systems using conventional pipelined broadcast algorithm, it can almost take advantage of the maximum bandwidth of transmission bus. Then, it was implemented with VerilogHDL, synthesized with TSMC 0.18um library and implemented into a chip. The area of synthesis results occupied 4,700 gates(2 input NAND gate) and utilization of total area is 2.4%. The proposed architecture achieves improvement in total performance of MPSoC occupying relatively small area.

※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 20120005728)
◆ 주저자 : 연세대학교 전기전자공학과 프로세서 연구실, hjyun@mpu.yonsei.ac.kr, 준회원
° 교신저자 : 연세대학교 전기전자공학과 프로세서 연구실, yonglee@yonsei.ac.kr, 종신회원
* 연세대학교 전기전자공학과 프로세서 연구실, wychung@mpu.yonsei.ac.kr, 정회원
논문번호 : KICS2011-12-631, 접수일자 : 2011년 12월 23일, 최종논문접수일자 : 2012년 7월 5일

Ⅰ. 서 론

최근 스마트폰(smart phone)과 같은 모바일 기기들의 영향으로 임베디드 시스템(embedded system)에서도 다양한 애플리케이션(application)과 그 연산의 복잡도가 증가하므로 고성능 프로세서 시스템에 대한 요구가 높아지고 있다. 하지만 단일 프로세서의 성능을 높이는 방법으로는 애플리케이션의 발전 속도를 따라가는데 한계가 있다. 하나의 칩 안에 다수의 프로세서와 메모리를 집적시키는 MPSoC (Multi-Processor System-on-Chip)는 단일 프로세서의 처리량을 여러 프로세서로 분산시킴으로써 시스템 전체의 성능을 향상 시킬 수 있다. 또한 전력 소모가 적기 때문에 임베디드 시스템 분야에서 연구가 활발히 진행되고 있다. 예로는 Apple의 A5, Texas Instruments의 OMAP, Nvidia의 Tegra, Samsung의 Exynos 등과 같은 SoC 플랫폼(System-on-Chip Platform)을 들 수 있다[1].

병렬 프로세서 간 데이터를 공유하는 방법은 크게 공유(shared) 메모리 방식과 분산(distributed) 메모리 방식으로 나뉜다. 공유 메모리 방식은 안전한 데이터 교환이 보장되고 프로그래밍이 쉽다는 장점이 있다[2]. 하지만 연결 노드의 개수가 늘어남에 따라 공유된 메모리에서 버스 트래픽 병목 현상에 의해 성능이 하락되고, 캐시 메모리의 일관성을 유지하기 위해 스눕(snoop) 오버헤드가 급격히 증가하는 단점을 가지고 있다[3,4]. 반면 분산 메모리 방식은 메시지 전달 방식을 통해 데이터를 공유하기 때문에 프로그래머가 상세히 지정해 줘야 하며, 데이터 전송 시 교착상태(deadlock)가 발생할 수 있다는 단점이 있다. 하지만 프로세서의 수가 늘어날수록 소비전력 및 수행 시간 등 데이터 전송 오버헤드가 작다는 장점이 있어 본 연구에서는 분산 메모리 구조를 사용하였다[5].

메시지 패싱(message passing)은 분산메모리 구조에서 가장 흔한 프로그래밍 모델로써 클러스터 기반의 시스템과 같은 고성능 컴퓨팅(HPC : High Performance Computing)에서 자주 사용된다[6]. 비록 메시지 패싱이 일반적으로 HPC에서 사용되고 있지만, 최근 임베디드 시스템에서의 FPGA(Field Programmable Gate Arrays)를 이용하여 MPI를 보다 효율적으로 구현할 수 있어 임베디드 애플리케이션(embedded application)에서의 관심이 높아지고 있다[7,8]. 또한 디지털 셋탑 박스(set-top box)와 모바일 기기에서 등에서 HD(High Definition) 급의 화질을 요구하기 때문에 임베디드 애플리케이션 분야에서의 관심이 높다[9].

MPI(Message Passing Interface)[10] 표준은 메시지 패싱 라이브러리를 위한 API(Application Programming Interface)를 제공하며, MPI 표준은 사실상 현재 메시지 패싱의 표준으로 사용되고 있다. MPI 표준은 다양한 점대점 통신(point-to-point communication), 집합 통신(collective communication)을 포함하고 있다. 이중 MPI 라이브러리에 정의된 집합 통신 함수는 MPI library cell에 의해 점대점 통신 함수들의 집합으로 변환되기 때문에 사용자에게 프로그래밍의 편리함을 제공하며, 이로 인해 사용자들이 자주 사용한다. 한 프로파일링 연구에선 특정 애플리케이션에서 전송 시간의 약 80%가 집합통신에 의한 것이라 명시하고 있다[11]. 이러한 집합 통신 함수 중에서 가장 흔히 사용되는 것 중 하나가 MPI_Bcast이다.

MPI_Bcast 루틴은 루트 노드의 메모리 데이터를 같은 커뮤니케이터(communicator) 내의 모든 노드의 메모리에 복사하는 통신 형태로써 일대다 통신에 속한다. 이러한 브로드캐스팅을 효과적으로 수행하기 위해 크게 3가지 형태로 연구가 진행되고 있다.

첫째로, MPI 브로드캐스트 동작을 효과적으로 수행하기 위한 연구로는 우선 브로드캐스트 알고리즘에 대한 연구가 있다. Sequential tree 알고리즘을 시작으로 binomial tree, binary tree, MST(Minimum Spanning Tree), distance MST, pipelined broadcast, Van de Gejin(hybrid) broadcast, modified Van de Gejin broadcast 등 많은 브로드캐스트 알고리즘이 제안되었다[12-15]. 이러한 알고리즘 중에서 성능 향상과 함께 구현이 용이한 알고리즘을 MPI library cell에 포함하여 알고리즘에 따라 점대점 통신으로 변환한다.

두 번째로 이러한 여러 알고리즘 중 수개의 알고리즘을 MPI library cell에 포함시키고, MPI 함수에 표현된 정보를 통해 알고리즘을 선택하는 연구가 진행 중이다[16,17]. MPI_Bcast 루틴은 다음과 같다.

MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

즉 "buffer"(버퍼의 시작 주소), "count"(버퍼 주소의 개수), "datatype"(데이터 종류)을 통해 데이터

메시지 크기, “root”를 통해 루트 노드의 위치, “comm”을 통해 커뮤니케이터에 속한 프로세스의 개수를 알 수 있고, 이러한 정보를 통해 알고리즘을 선택한다.

마지막으로 MPI 통신을 수행하는 하드웨어를 설계하여 점대점 통신을 가속화하는 연구도 진행 중이다[18,19]. 이와 같이 MPI 통신을 지원하는 하드웨어를 사용할 경우, 소프트웨어적으로 처리할 때보다 처리 속도가 빠르며, MPI 통신을 하드웨어가 전담함 으로써 프로세서의 CPI(Clock per Instruction)를 올릴 수 있다.

현재의 브로드캐스트 알고리즘 중 파이프라인 브로드캐스트(pipelilned broadcast) 알고리즘은 데이터를 조각으로 나누어서 전송한다. 데이터를 조각으로 나누어 파이프라인의 형태로 메시지가 전송되기 때문에 대부분의 프로세싱 노드의 통신포트를 활성화 시킨다. 이에 다른 알고리즘에 비해 성능이 올라간다. 하지만 파이프라인 브로드캐스트 알고리즘의 경우도 나누어진 만큼 동기화 신호도 추가되므로 손실이 발생한다. 따라서 본 연구에서는 파이프라인 브로드캐스트 구조를 손실 없이 실행하는 알고리즘과 하드웨어인 파이프라인 체인(pipeline chain) 구조를 제안하였고, 이를 적용한 MPI 하드웨어 유닛을 SystemC를 이용하여 설계하였다. 또한 verilogHDL로 하드웨어를 설계하여 TSMC 0.18 라이브러리에서 합성하여 칩으로 제작 하였다.

Ⅱ. 기존연구

파이프라인 브로드캐스트 알고리즘은 MPI 브로드캐스트 통신을 가속화하기 위한 알고리즘이다. 파이프라인 브로드캐스트 알고리즘은 4장의 결과를 보면 데이터의 전송사이즈가 클수록 좋은 성능을 보여준다. 하지만 파이프라인 브로드캐스트 알고리즘의 구조 때문에 버스의 전송 대역폭을 충분히 활용하지 못하는 단점이 있다. 1절 에서는 파이프라인 브로드캐스트 알고리즘을 설명하고, 2절 에서는 파이프라인 브로드캐스트 알고리즘의 문제점에 대하여 설명한다.

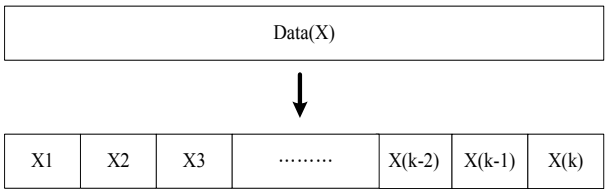


그림 1. k개로 데이터 분할
Fig. 1. Data partitioning into k

2.1. 파이프라인 브로드캐스트 알고리즘

파이프라인 브로드캐스트 알고리즘[14][15]은 브로드캐스트 통신에 사용할 데이터를 그림 1처럼 k개로 나누어서 체인 형태로 그림 2와 같이 전송하는 방식이다. 데이터를 나누어서 전송하기 때문에 그림 2를 보면 스텝3부터 스텝k까지 브로드캐스팅에 필요한 모든 통신채널을 사용하므로, 전송대역폭을 최대한으로 사용하는 것을 확인할 수 있다. Sequential tree, Binomial tree 등 다른 알고리즘은 브로드캐스트에 필요한 통신채널을 모두 활용할 수 없기 때문에 전송대역폭을 모두 활용하지 못한다.

파이프라인 브로드캐스트는 그림 1처럼 k개로 데이터를 분할하여 데이터를 전송 하는데, 이 때 사용되는 최적화 된 k 값을 사용하는 것은 중요하다. k 값이 너무 작으면 불필요한 동기화 과정은 줄어들지만, 모든 통신포트가 활성화 되는 부분이 작게 발생하여 전송대역폭을 충분히 활용할 수 없다. 하지만 k 값이 너무 크면 모든 통신가 활성화 되는 부분이 많이 발생하여 전송대역폭을 충분히 활용할 수 있지만 불필요한 동기화 과정이 많이 발생하여 손실이 발생한다. 따라서 이전 연구에서 최적화 된 k값을 구했었고[14,15], 식 (1)과 같다.

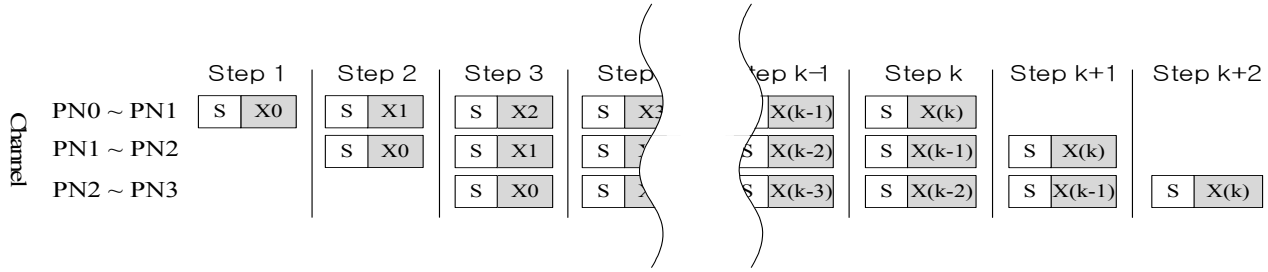


그림 2. 파이프라인 브로드캐스트 구조의 통신과정
Fig. 2. The communication process of pipelined broadcast architecture

$$k_{opt}\begin{cases}1 & \text{if } \sqrt{(p-2)n\beta/\alpha} < 1 \\ n & \text{if } \sqrt{(p-2)n\beta/\alpha} > n \\ \sqrt{(p-2)n\beta/\alpha} & otherwise\end{cases} \quad (1)$$

위의 식에서 α는 동기화 수행 사이클이고, β는 1개의 데이터 전송 사이클이고, p는 전체 프로세싱 노드의 수이고, n은 전송할 데이터 이다.

2.2. 기존연구의 문제점

파이프라인 브로드캐스트 알고리즘은 데이터를 전송 시 데이터를 k개로 나누어 메시지를 그림 2와 같이 전송하므로 높은 성능으로 전송한다. 하지만 파이프라인 브로드캐스트 알고리즘의 과정을 더 자세히 보면 매 단계(step) 마다 동기화(S) 과정이 k번만큼 중복되는 것을 볼 수 있다.

제안하는 구조와 실험에서 사용하였던 모델의 구조는 동기화 시간이 16 사이클 이므로, 전체 브로드캐스트 전송에서 동기화 시간이 k*16*10ns(100MHz의 동작주파수) 이 되고, k 값이 커진다면 전체 시스템에 많은 손실을 발생시킬 것이다.

브로드캐스트 통신의 경우 커뮤니케이터내의 모든 프로세싱 노드들에게 공유 데이터를 전송하기 때문에, 모든 프로세싱 노드들의 통신포트를 사용할 것이다. 따라서 데이터를 전송하기 전에 사용할 모든 통신포트들을 할당하고 파이프라인으로 데이터를 전송하여 준다면 가장 좋은 성능을 보여줄 것이다. 또한 이러한 방식은 서킷 스위칭과 유사한 데이터 전송 구조와 비슷하다.

2.2.1. 서킷 스위칭(Circuit Switching)

서킷 스위칭[20]은 송신부와 수신부의 채널을 우선 할당하고, 버퍼없이 데이터를 전송하는 구조이다. 그림 3-2는 서킷 스위칭 통신 과정이다. 그림 3에서 시간은 왼쪽에서 오른쪽으로 흐르고, R은 요청(request), A은 응답(acknowledgement), D은 데이터(data), T는 종료(Tail)을 나타낸다. 그림 3을 보면 통신에 필요한 모든 채널을 할당하기 위하여 요청과 응답 메시지를 통신하는 것을 볼 수 있다. 또한 이 과정이 끝난 후에 할당된 채널을 통하여 데이터를 전송하고, 종료 메시지로 할당된 채널을 반환하는 것을 확인할 수 있다.

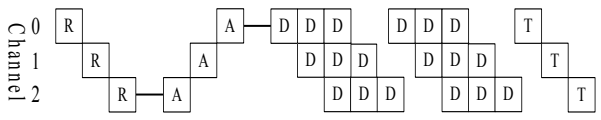


그림 3. 서킷 스위칭 구조의 통신과정
Fig. 3. The communication process of circuit switching architecture

2.2.2. Sequential Tree 알고리즘

그림 4와 같이 Sequential Tree 알고리즘은 루트 프로세싱노드에서 나머지 프로세싱노드에게 데이터를 순차적으로 전송한다. 총 N 개의 프로세싱노드가 있다면 Sequential Tree 알고리즘에서 루트 노드는 그림 4와 같이 N-1번 데이터전송을 한다.

Ⅲ. 제안하는 알고리즘 및 MPI 하드웨어 유닛 구조

파이프라인 브로드캐스트 알고리즘의 경우 최적화 된 k 값을 사용하여 데이터를 나누어 전송하여도, 동기화 과정이 k 만큼 반복되기 때문에 시스템 성능의 손실이 발생한다. 이에 본 논문에서는 동기화 과정의 중복이 없는 서킷 스위칭(circuit-switching) 기반[20]의 파이프라인 체인 알고리즘 및 하드웨어 구조를 제안한다. 1 절에서 파이프라인 체인 알고리즘을 설명하고, 2 절에서는 이를 실행하는 하드웨어 구조를 설명한다.

3.1. 파이프라인 체인 알고리즘

이전에 연구되었던 파이프라인 브로드캐스트 알고리즘은 그림 2와 같이 데이터를 k개로 나누어 전송을 한다. 또 한 브로드캐스트 알고리즘의 전송되는 과정을 보면 머리(head), 몸체(body), 꼬리(tail) 노드로 나누어진다. 그림 2의 경우 MPI library cell을 사용하여 점대점 통신으로 변환하는 컴파일러에서 파이프라인 알고리즘을 사용할 경우 머리노드는 노드 0 이고, 몸체 노드는 노드 1와 2, 꼬리 노드는 노드 3이다. 머리 노드의 경우 몸체노드로 데이터를 전송하기 때문에 MPI_Bcast 명령이 k개의 MPI_Send 명령으로, 꼬리노드의 경우 몸체노드로부

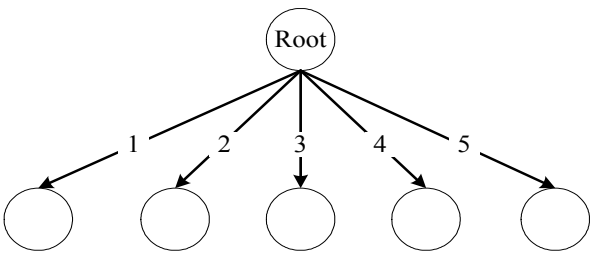


그림 4. Sequential Tree 알고리즘
Fig. 4. Sequential Tree Algorithm

터 데이터를 수신 받기 때문에 k개의 MPI_Recv 명령으로 변환된다. 몸체 노드는 데이터를 다른 노드로부터 수신하면서 전송하여야 하기 때문에 k개의 MPI_Send 와 MPI_Recv 명령으로 변환된다. 그리고 MPI 유닛은 이러한 순서로 통신을 실행한다. 하지만 각 노드들에서 k 만큼의 점대점 명령어로 변환되므로 k 만큼의 동기화 신호가 중복되어 손실이 발생한다. 만약 파이프라인 브로드캐스트 알고리즘을 그림 6처럼 동기화 과정을 반복하지 않고, 데이터를 그림 5처럼 워드단위로 나누어 전송한다면, 데이터를 전송하는 동안 각 노드에서 활성화 되는 통신포트를 최대로 사용할 수 있고, 이는 전송대역폭을 최대로 활용하여 데이터를 전송할 것이다. 따라서 본 논문에서는 파이프라인 체인 알고리즘을 제안한다.

본 논문에서 제안하는 파이프라인 체인 (Pipeline-Chain) 알고리즘은 서킷 스위칭(Circuit Switching)[20] 구조를 MPI 통신에서 사용할 수 있도록 수정하고, MPI 하드웨어 구조를 최적화 하였다. 그림 6은 파이프라인 체인 알고리즘의 구조이다. 파이프라인 체인 알고리즘은 데이터를 동기화 과정의 중복 없이 워드단위로 전송하기 위하여, 전송에 앞서 모든 데이터 채널을 할당하고 통신한다. 그림 6에서 시간은 좌에서 우로 흐르며, S는 채널을 할당하는 동기화과정이다. 총 3개의 통신 채널들이 동작하는 것을 보여준다. MPI 통신에서 브로드캐스트 통신은 루트노드가 다른 모든 노드들에게 데이터를 전송하기 때문에, 모든 노드에서 통신채널을 할당받아 데이터를 파이프라인으로 전송한다면

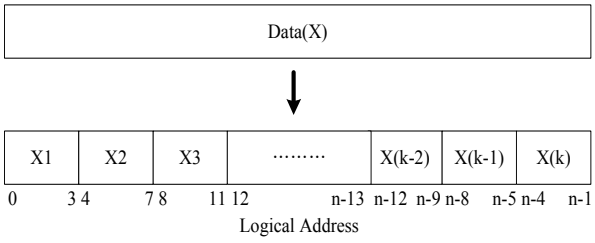


그림 5. 워드로의 데이터 분할
Fig. 5. Data partitioning into word

전송대역폭을 최대로 활용하여 가장 좋은 성능을 보여줄 것이다.

서킷 스위칭 기반의 파이프라인 체인 알고리즘은 이러한 손실을 줄이기 위하여 그림 5와 같이 데이터를 워드 단위로 나누어 동기화 과정의 중복 없이 전송하는 구조이다.(본 구조에서 워드는 4bytes 이다.) 이와 같이 제안하는 브로드캐스트 알고리즘이 적용되기 위해서 컴파일러는 MPI_Bcast 루틴을 워드단위의 데이터로 나누어 점대점 통신으로 변환하지 않고, 동기화 신호 없이 나누어진 데이터 워드를 버퍼에 저장하면서 다른 노드로 전송하여 주는 명령어와 하드웨어 구조가 필요하다. 이에 제안하는 알고리즘을 위하여 MPI_Fwd 루틴이 추가되었다. 따라서 본 논문에서는 user level 의 MPI library cell에서 MPI_Bcast 루틴을 머리노드의 경우 MPI_Send 명령으로, 몸체노드의 경우 MPI_Fwd 명령으로, 꼬리노드의 경우 MPI_Recv 명령으로 MPI 유닛에 전달된다.

3.2. 파이프라인 체인 하드웨어 구조

그림 7은 파이프라인 체인 알고리즘을 동작하기 위하여 본 논문에서 제안하는 MPI 유닛의 전체 구조이다. 프로세서는 명령어 메모리(instruction memory)의 명령어 중 MPI 명령을 MPI 유닛에 맞추어 변환하는 역할을 하고, 메모리 컨트롤러(memory controller)는 데이터 메모리(Data Memory)에서 MPI 통신에 필요한 데이터를 읽고 쓰는 역할을 한다. 마스터 래퍼(master wrapper)와 슬래이브 래퍼(slave wrapper)는 MPI BUS와 MPI 유닛과의 인터페이스 역할을 한다.

제안하는 MPI 유닛의 구조는 MPI 유닛에서 파이프라인-체인 브로드캐스트 통신을 처리하기 위하여 메시지 프로세싱 엔진(message processing engine)의 내부구조가 변경되었다. 메시지 프로세싱 유닛 에는 MPI_Fwd 명령을 처리하기 위한 하드웨어가 추가되었고, 또한 본 구조에서는 MPI_Fwd 명령을 처리하기 위하여 송신과 수신이 동시에 처리

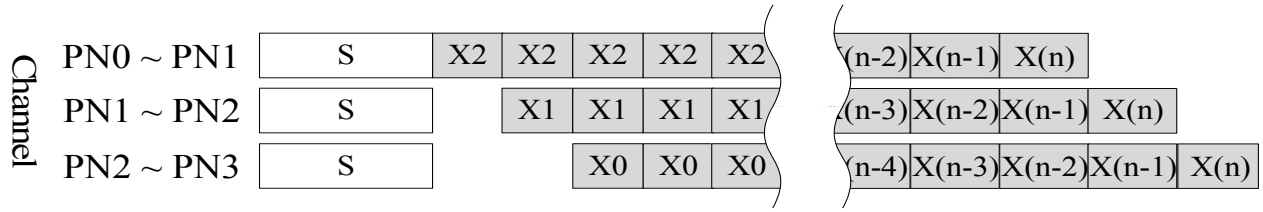


그림 6. 파이프라인 체인 구조의 통신과정
Fig. 6. The communication process of pipeline chain architecture

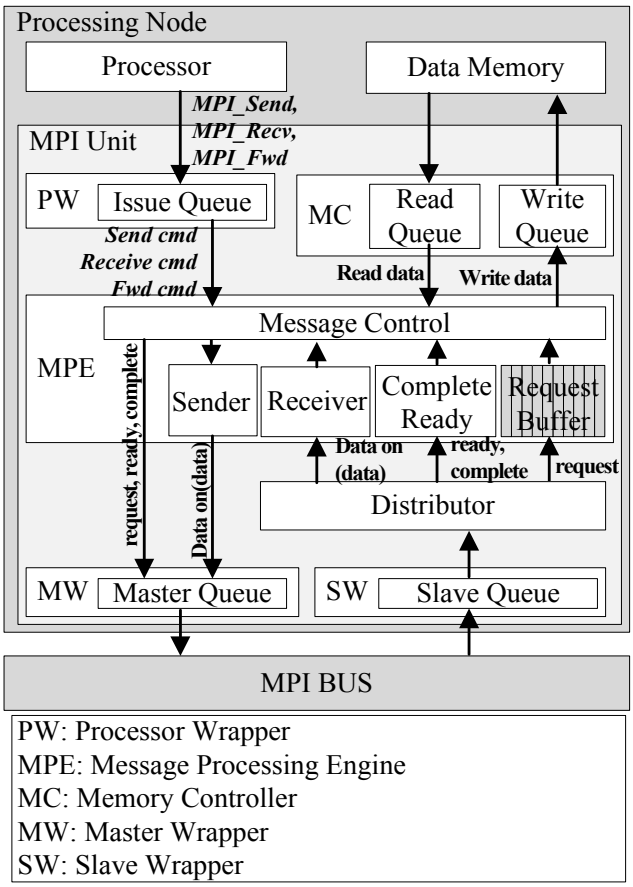


그림 7. 제안하는 구조의 프로세싱 노드
Fig. 7. The processing node of proposed architecture

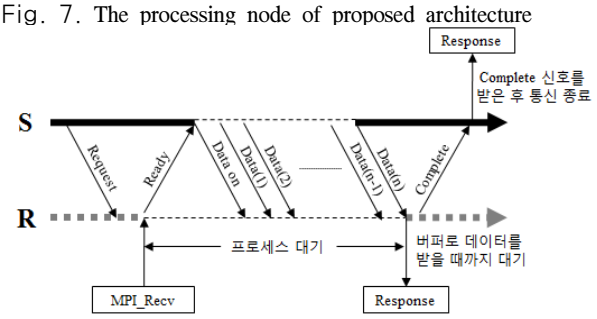


그림 8. 동기송신 과정
Fig. 8. The process of synchronous send

되어야 함으로 메시지 프로세싱 엔진내의 송신부(sender)와 수신부(recver)를 분리하였다.

3.2.1. 분배기(Distributor)

분배기는 슬래이브 래퍼(slave wrapper)를 통하여 MPI BUS에서 들어오는 요청 메시지를 분리하여 메시지 프로세싱 유닛의 수신버퍼에 저장한다. MPI 통신 프로세싱 노드로는 데이터 값, 요청(request), 준비(request)와 완료(complete) 메시지가 수신된다.

3.2.2. 메시지 프로세싱 엔진(Message Processing Engine)

브로드캐스트 통신은 루트노드의 데이터를 같은 커뮤니케이터(communicator) 내의 모든 노드에게 전송한다. 따라서 브로드캐스트 체인 알고리즘을 이용하여 전송한다면, MPI BUS에서 높은 대역폭으로 전송할 수 있다. 메시지 프로세싱 엔진을 사용한 통신의 방법은 아래와 같다.

메시지 프로세싱 엔진에서 처리하는 통신 명령은 총 3가지(MPI_Send, MPI_Recv, MPI_Fwd)가 있고, 또한 파이프라인 체인은 3가지(머리, 몸체, 꼬리) 노드의 송수신으로 나뉜다. 동기송신을 사용할 경우 머리노드와 꼬리노드의 MPI_Send(head), MPI_Recv(tail) 와 같다. 그림 8의 동기송신 과정에서 머리노드의 경우 S는 머리노드, R은 i+1노드(i=자신의 노드번호), 꼬리노드의 경우 S는 i-1노드(i=자신의 노드번호) R은 꼬리노드 인 경우를 제외하고 동기송신 과정과 같다.

먼저 쌍이 되는 MPI_Send 와 MPI_Recv 명령의 동작은 다음과 같다. MPI_Send 명령이 송신노드의 메시지 프로세싱 엔진으로 들어오게 되면, 그림 8의 동작을 메시지 프로세싱 엔진에서 실행한다. 먼저 메시지 프로세싱 엔진에서는 요청(request) 메시지를 마스터 래퍼(master wrapper)를 이용하여 버스에 전

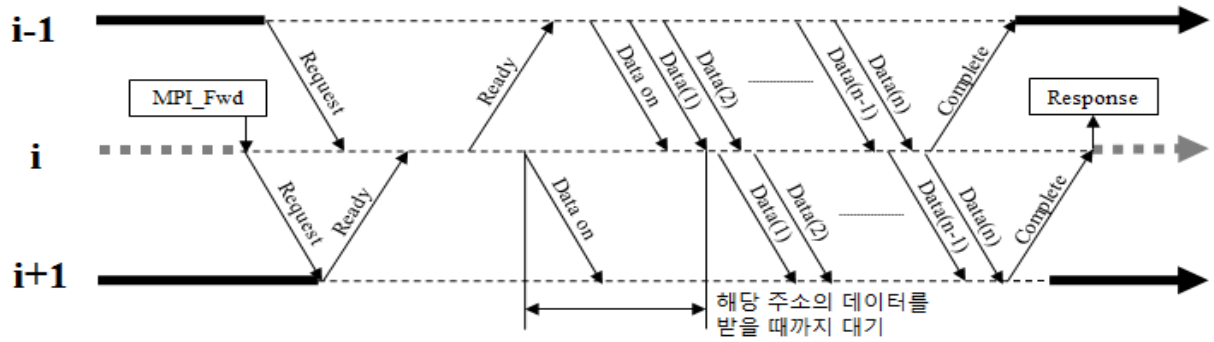


그림 9. MPI_Fwd 명령 흐름
Fig. 9. The Flow of MPI_Fwd command

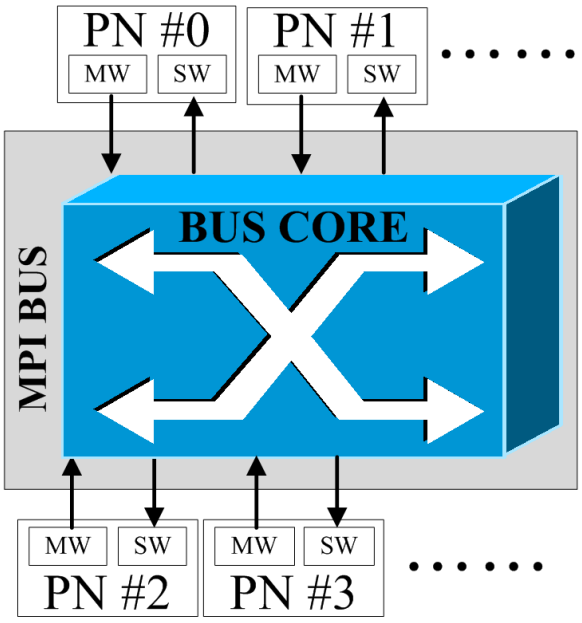


그림 10. MPI 버스 구조
Fig. 10. The architecture of MPI BUS

송하고, 수신 노드에서는 메시지 프로세싱 엔진안의 수신버퍼로 저장한다. 송신노드에서는 요청 메시지를 전송 후 준비(ready) 메시지를 기다린다. 송신노드가 준비 메시지를 받으면 데이터온(data-on) 메시지를 시작으로 송신기(Sender)를 이용하여 데이터를 전송한다. 송신노드에서는 데이터 전송을 완료 후, 완료(complete) 메시지를 기다린다. 송신노드에서 완료 메시지를 받으면 MPI_Send 명령이 종료된다.

MPI_Recv 명령의 동작은 아래와 같다. MPI_Recv 명령이 수신 노드의 메시지 프로세싱 엔진으로 들어오면, 엔진내의 수신버퍼에서 MPI_Recv 명령과 쌍이 되는 메시지를 찾는다. 메시지 프로세싱 엔진내의 수신버퍼에서 MPI_Recv 와 쌍이 되는 메시지를 찾으면 송신노드로 마스터 래퍼를 이용하여 준비 메시지를 전송하고, 데이터온 메시지와 데이터를 수신기(Receiver)를 이용하여 받을 준비를 한다. 데이터 전송완료 후 송신노드로 마스터 래퍼를 이용하여 완료 메시지를 보낸 후 MPI_Recv 명령이 끝나게 된다.

MPI_Fwd 는 파이프라인-체인 구조를 위한 명령이고 동작은 그림 9와 같다. 그림 9는 메시지 프로세싱 엔진에서 MPI_Fwd 명령을 받고 파이프라인-체인 브로드캐스트 알고리즘의 몸체(body)노드로써의 동작 흐름도이고 아래와 같다.

만약 위의 순서로 하지 않는다면 메시지 프로세싱 엔진내의 송신기와 수신기를 동시에 사용하지 못하여 교착(deadlock) 상태가 발생한다.

① i+1노드(i=자신의 노드번호)로 요청(request) 메시지 송신
② i-1노드(i=자신의 노드번호)로부터 요청 메시지 수신
③ i-1노드로 준비(ready) 메시지 송신
④ i+1노드로부터 준비 메시지 수신
⑤ 송신기(sender)와 수신기(Recver)를 이용하여 인접노드와 데이터 송·수신
⑥ 인접노드와 완료(complete) 메시지 송·수신

3.2.3. MPI 버스

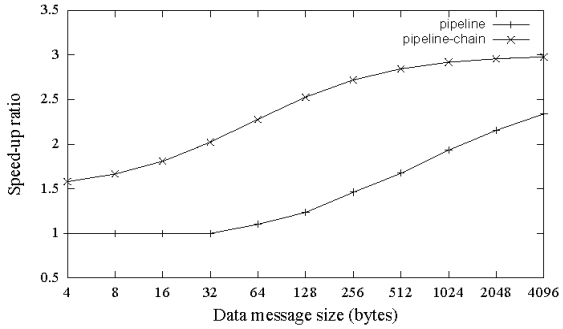
MPI 버스는 커뮤니케이터에서 사용하는 버스구조이며, 그림 10과 같다. 그림 10과 같이 MPI 버스는 크로스 바(Cross Bar) 구조이며, 커뮤니케이터내의 모든 프로세싱 노드들이 MPI 버스를 통하여 데이터를 전송하는 구조이다. 각각의 프로세싱 노드들은 노드 내의 마스터 래퍼(MW)를 통하여 데이터를 전송하고, 슬레이브 래퍼(SW)를 통하여 데이터를 받는다.

MPI 버스는 크로스바(Cross bar) 버스인 AMBA AXI 버스를 MPI 통신에 맞게 최적화한 구조이다. 제한된 칩의 면적으로 인하여 AMBA AXI 버스에서 MPI 데이터 통신에 필요한 일부분만 구현하여 설계하였다.

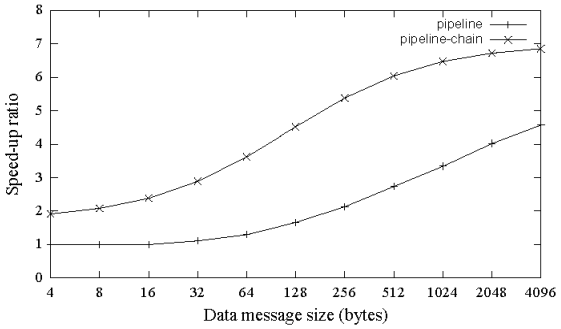
Ⅳ. BFM 시뮬레이션 및 결과

제안하는 MPI 유닛의 성능을 측정하기 위해 이전에 연구되었던 모델과 제안하는 모델을 비교하였다. 이전에 연구 되었던 모델은 브로드캐스트 통신에서 데이터를 k개로 나누어서 통신을 하는 파이프라인 브로드캐스트 모델이고, 제안하는 모델(파이프라인 체인)이다. 각 모델은 systemC를 사용하여 BFM(Bus Functional Model)을 설계하였다. BFM은 각 블록의 지연시간을 고려하여 동작을 기술하였고, 특정 시뮬레이션 환경에 따라 통신 트래픽을 생성할 수 있다. 그 후 생성된 통신 트래픽으로 제안한 MPI 유닛의 전송 시간을 확인하였다.

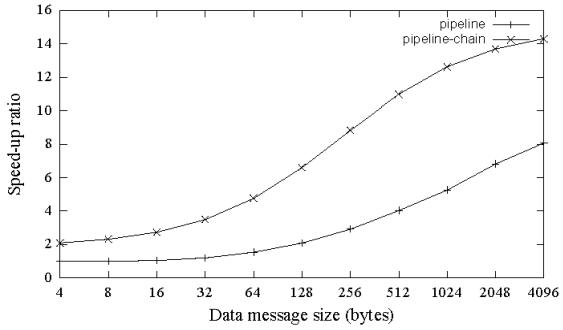
여러 상황에서 성능을 평가하기 위하여 한 커뮤니케이터 내의 프로세싱 노드의 수를 4, 8, 16, 32개로 나누었으며, 브로드캐스트 통신의 데이터 사이즈를 임베디드 시스템에서 자주 사용되는 4Byte ~ 4KByte 로 나누어서 sequential tree, 파이프라인 브로드캐스트, 제안하는 구조(파이프라인 체인)을 실험



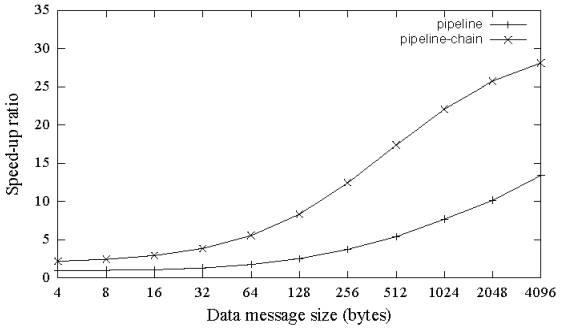
(a). Total 4-processing node
(a). 총 4개의 프로세싱 노드



(b). Total 8-processing node
(b). 총 8개의 프로세싱 노드



(c). Total 16-processing node
(c). 총 16개의 프로세싱 노드



(d). Total 32-processing node
(d). 총 32개의 프로세싱 노드

그림 11. Sequential tree 알고리즘과 비교한 실험 결과
Fig. 11. The simulation results comparing with sequential tree algorithm

하여 비교하였다.

그림 11-(a)는 4개의 프로세싱 노드를 사용하여 브로드캐스트 통신의 성능을 평가한 것이다. 가로축은 브로드캐스트 데이터 사이즈이고, 세로축은 브로드캐스트 통신에 걸린 시간이 sequential tree 알고리즘 모델에 비해 이전에 연구되었던 모델(파이프라인 브로드캐스트)과 제안하는 모델(파이프라인-체인 브로드캐스트)이 빨라진 정도(speed-up ratio) 이다. k-1 만큼 반복되는 동기화 과정이 줄었고, 데이터가 동기화 과정의 중복으로 인한 손실 없이 워드 단위로 나뉘어 전송을 함으로, 전체적으로 제안하는 모델이 기존의 모델보다 성능이 향상된 것을 복수 있다.

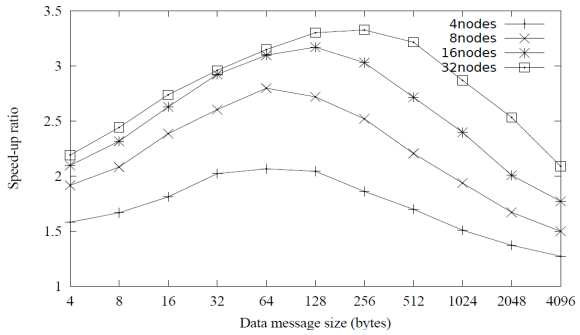


그림 12. 파이프라인 브로드캐스트 알고리즘과 비교한 실험 결과
Fig. 12. The simulation results comparing pipelined broadcast algorithm

그림 11-(b)는 8개의 프로세싱 노드를, 그림 11-(c)는 16개의 프로세싱 노드를, 그림 11-(d)는 32개의 프로세싱 노드를 사용하여 브로드캐스트 통신의 성능을 평가한 것이다. 8, 16, 32개의 프로세싱 노드로 구성된 시스템에서도 같은 이유로 성능이 향상된 것을 볼수 있다.

그림 12는 4, 8, 16, 32개의 노드로 사용한 시스템에서 이전 연구되었던 모델(파이프라인 브로드캐스트)과 제안하는 모델(파이프라인-체인 브로드캐스트)의 브로드캐스트 통신의 빨라진 정도(speed-up ratio)로 비교한 것이고, 최대 3.3배의 성능향상이 있었다.

V. 하드웨어 설계

제안하는 MPI 하드웨어 유닛은 처음 SystemC를 사용하여 BFM(Bus Functional Model)하여, 총 프로세싱 노드수, 브로드캐스트 전송 데이터 크기 등

과 같은 파라미터들을 변화 하여 성능을 확인하였다. 그 후 VerilogHDL 언어를 이용하여 4개의 프로세싱 노드(processing node)를 포함한 하나의 커뮤니케이터(communicator)를 설계하여 실제 칩으로 제작하였다. 칩 제작을 위하여 TSMC 0.18um 의 표준 셀 라이브러리를 사용하여 synopsys사의 design compiler를 이용하여 합성하였고, 차후에 synopsys사의 ASTRO를 사용하여 레이아웃(layout) 하였다.

표 1은 제안하는 구조의 합성 결과이다. 표 1을 보면 설계한 모듈을 계층별로 면적(area)과 그에 대한 비율을 보여준다. 칩 내의 MPI 커뮤니케이터는 3x3mm 의 다이 사이즈(die size)의 제한으로 인하여 총 4개의 프로세싱 노드(p0, p1, p2, p3)와 MPI 버스(mpi_bus)만 집적할 수 있었다. 프로세싱 노드는 YMPU(ympu), MPI 유닛(mpi), 각 4Kbit의 명령어(i_mem)와 데이터(d_mem)메모리로 이루어져 있다. MPI 유닛은 4700 게이트(2-input NAND 기준) 정도의 면적과 전체 면적대비 비율이 2.4%(0.6x4) 이므로 칩 사이즈에 비해 무시해도 좋을 만하다. 이에 본 논문에서 설계한 MPI 유닛은 작은 면적으로 전체 시스템의 성능을 높이는데 유용하다.

그림 13은 제안하는 구조의 레이아웃 사진이다. 칩 내의 8개의 블록은 4개의 YMPU에서 사용하는 메모리(명령어, 데이터) 블록을 보여 주고 있으며,

표 1. 제안하는 유닛의 계층별 합성 결과
Table 1. The hierarchical synthesis result of proposed architecture

hierarchy			area	utilization
mpi_comm			185914.65	100.0%
	mpi_bus		2429.26	1.3%
	p0		45907.74	24.7%
		ympu	28281.60	15.2%
		mpi	1189.85	0.6%
	p1		45781.82	24.6%
		ympu	28164.52	15.1%
		mpi	1180.54	0.6%
	p2		45804.47	24.6%
		ympu	28189.49	15.2%
		mpi	1178.21	0.6%
	p3		45794.37	24.6%
		ympu	28167.18	15.2%
		mpi	1190.52	0.6%

전체 적인 레이아웃 과정은 오토 레이아웃(auto place & routing) 기능을 사용하였다. ASTRO에서 레이아웃시 모든 모듈이 언 그룹(ungroup)되어 레이아웃 되기 때문에 블록으로 나눠지지 않았다.

Ⅵ. 결 론

MPSoC는 처리 속도의 향상, 설계의 유연성, 저전력 소비, 설계시간 단축 등 많은 이점이 있어 임베디드 시스템에서 최근 연구가 활발하다. 애플리케이션의 증가와 그에 따른 연산량의 증가로 프로세서의 수는 계속적으로 증가되는 추세지만, 태스크 분할 문제, 통신 오버헤드로 인한 병목현상 등으로 인해 선형적으로 성능향상이 되지 않는다. 따라서 본 논문에선 멀티프로세서 시스템에서 브로드캐스트 통신의 전송시간을 줄이기 위해, 전송 버스의 대역폭을 최대한으로 활용하는 알고리즘과 MPI 유닛을 제안하고 설계하였다. MPI 유닛 내부의 메시지 프로세싱 엔진에서는 MPI_Fwd 명령이 추가되었고, 이를 사용하여 파이프라인 체인 알고리즘을 실행함으로 메시지 전달 효율을 높였다.

제안하는 파이프라인 체인 구조는 Sequential Tree 알고리즘과 비교하면 최대 N_NODE-1(N_NODE=총 프로세싱 노드의 수)의 성능향상이 있었다. 또한 파이프라인 브로드캐스트 구조와 비교하면 최대 3.3배의 성능향상이 있었다. 또한 BFM 시뮬레이션 후, VerilogHDL 언어를 이

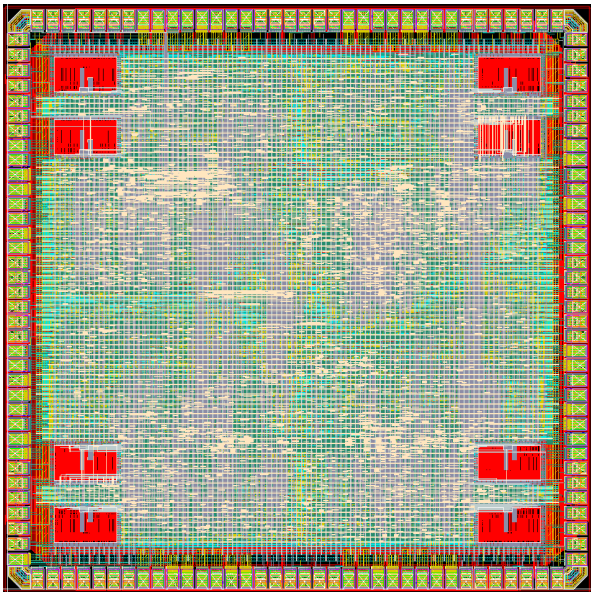


그림 13. 제안하는 유닛의 레이아웃 결과
Fig. 13. The layout result of proposed architecture

용하여 하드웨어 구조를 기술하였고, 이렇게 기술된 하드웨어는 synopsys사의 design compiler를 사용하여 합성하였다. 합성 라이브러리는 TSMC 공정의 0.18um 공정 라이브러리를 사용해 합성하였고 100Mhz에서 게이트 레벨 시뮬레이션이 동작하였고, MPW를 통해 실제 칩으로 제작하였다.

본 논문에서 제안하고 설계한 MPI유닛을 사용하면 전송 대역폭을 극대화 할 수 있기에 분산 메모리 아키텍처를 사용하는 멀티프로세서 시스템에서 매우 효과적이다.

References

[1] L. Gwennap, *"Apple A5 Adds New Features," Microprocessor report*, May 2011

[2] A. C. Klaiber, H. M. Levy, *"A comparison of message passing and shared memory architectures for data parallel programs," Proceedings of the 21st annual international symposium on Computer architecture*, Vol 22, Apr. 1994, pp. 94-105

[3] P. Stenstrom, *"A Survey of Cache Coherence Schemes for Multiprocessors,"* Computer, Vol. 23, Jun. 1990, pp. 12-24.

[4] M. Tomasevic and V.M. Milutinovic, *"Hardware Approaches to Cache Coherence in Shared-Memory Multiprocessors," IEEE Micro*, vol. 14, nos. 5-6, Oct./Dec. 1994, pp. 52-59.

[5] L. Benini and G.de Micheli, *" Networks On Chip: A New SoC Paradigm," IEEE Computer*, Vol 35, No. 1, Jan. 2002, pp. 70-78.

[6] Daniel L. Ly, Manuel Saldana, Paul Chow, "*The Challenges of Using An Embedded MPI for Hardware-based Processing Nodes," Field-Programmable Technology(FPT)* 2009, Sydney, NSW, Dec. 2009, pp. 120-127.

[7] T. P. McMahon and A. Skjellum, "*eMPI/eMPICH: Embedding MPI," MPI Developers Conf.*, 1996, pp. 180-184.

[8] M. Saldana, A. Patel, C. Madill, N. D., A. Wang, A. Putnam, R. Wittig, and P. Chow, "*MPI as an abstraction for software-hardware interaction for HPRCs," in International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, Nov. 2008, pp. 1 – 10.

[9] C. Pedraza, E. Castillo, J. Castillo, C. Camarero, J. Bosque, J. Martinez, and R. Menendez, "*Cluster architecture based on low cost reconfigurable hardware," in International Conference on Field Programmable Logic and Applications*, Sept. 2008, pp. 595 – 598.

[10] MPI-forum, "Message passing interface forum," Jan. 2009, uRL: http://www.mpi-forum.org.

[11] R. Rabenseifner, "*Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512," Proceedings of the Message Passing Interface Developer's and User's Conference 1999(MPIDC99)*, 1999, pp 77-85.

[12] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. *"Automatically Tuned Collective Communications," In Proc. of SC'00: High Performance Networking and Computing*, 2000.

[13] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, and Robert van de Geijn, "*Building a High-Performance Collective Communication Library," Supercomputing'94*, Nov. 1994, pp 107-116.

[14] M. Barnett, D. Payne, R. van de Geijn and J. Watts, "Broadcasting on meshes with worm-hole routing," Technical Report , Department of Computer Sciences, the University of Texas at Austin, Nov. 1994.

[15] J. Watts and R. van de Geijn, "*A Pipelined Broadcast for Multidimensional Meshes," Parallel Proc. Letter*, 1995

[16] Thakur, Rajeev, et al., "*Optimization of collective communication operations in mpich," International Journal of High Performance Computing Applications*, Feb. 2005, pp. 49-66.

[17] George Almasi, Charles J.Archer, C. Chris Erway, Philip Heidelberger, Xavier Martorell, Jose E. Moreira, B.Steinmacher-Burow, and YiliZheng, "*Optimization of MPI Collective Communication on BlueGene/L Systems," ICS'05, Prec. of the 19th annual international conf. on Supercomputing*, June. 2005, pp. 253-262.

[18] Poletti Francesco, Poggiali Antonio, and Paul Marchal, "*Flexible hardware/software support*

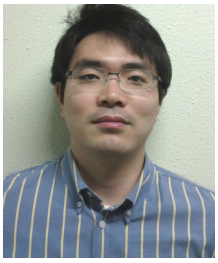
for message passing on a distributed shared memory architecture," Design, Automation and Test in Europe 2005, Mar., Vol. 2, 2005, pp. 736-741.

[19] Manuel Saldana and Paul Chow, "*TMD-MPI IMPLEMETATION FOR MULTIPLE PROCESSORS ACROSS MULTIPLE FPGAS," FPL'06*, Aug. 2006, pp. 1-6.

[20] W. J. Dally and B. Towles, Principles and Practices of Interconnection Networks. San Francisco, CA: Morgan Kaufmann, 2004

윤 희 준 (Heejun Yun)

2010년 숭실대학교 정보통신전자공학부 학사 졸업



2012년 연세대학교 전기전자공학과 석사 졸업

2012년~현재 LG전자 연구원

<관심분야> 컴퓨터 아키텍처, ASIC, SoC, AP

정 원 영 (Wonyoung Chung)

2005년 연세대학교 전기전자공학과 학사 졸업

2012년 연세대학교 전기전자공학과 박사 졸업

2012년~현재 삼성전자 책임연구원

<관심분야> 네트워크 프로세서, 컴퓨터 아키텍처, 메모리 구조

이 용 석 (Yong-surk Lee)

1973년 2월 연세대학교 전기공학과 졸업

1977년 2월 Michigan 반도체 설계 공학석사

1981년 3월 Michigan 반도체 설계 Ph.D

1993년 2월~현재 연세대학교 전기전자공학과 교수

<관심분야> 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계